

Serverless II

CNAE – Cloud Native Architecture & Engineering



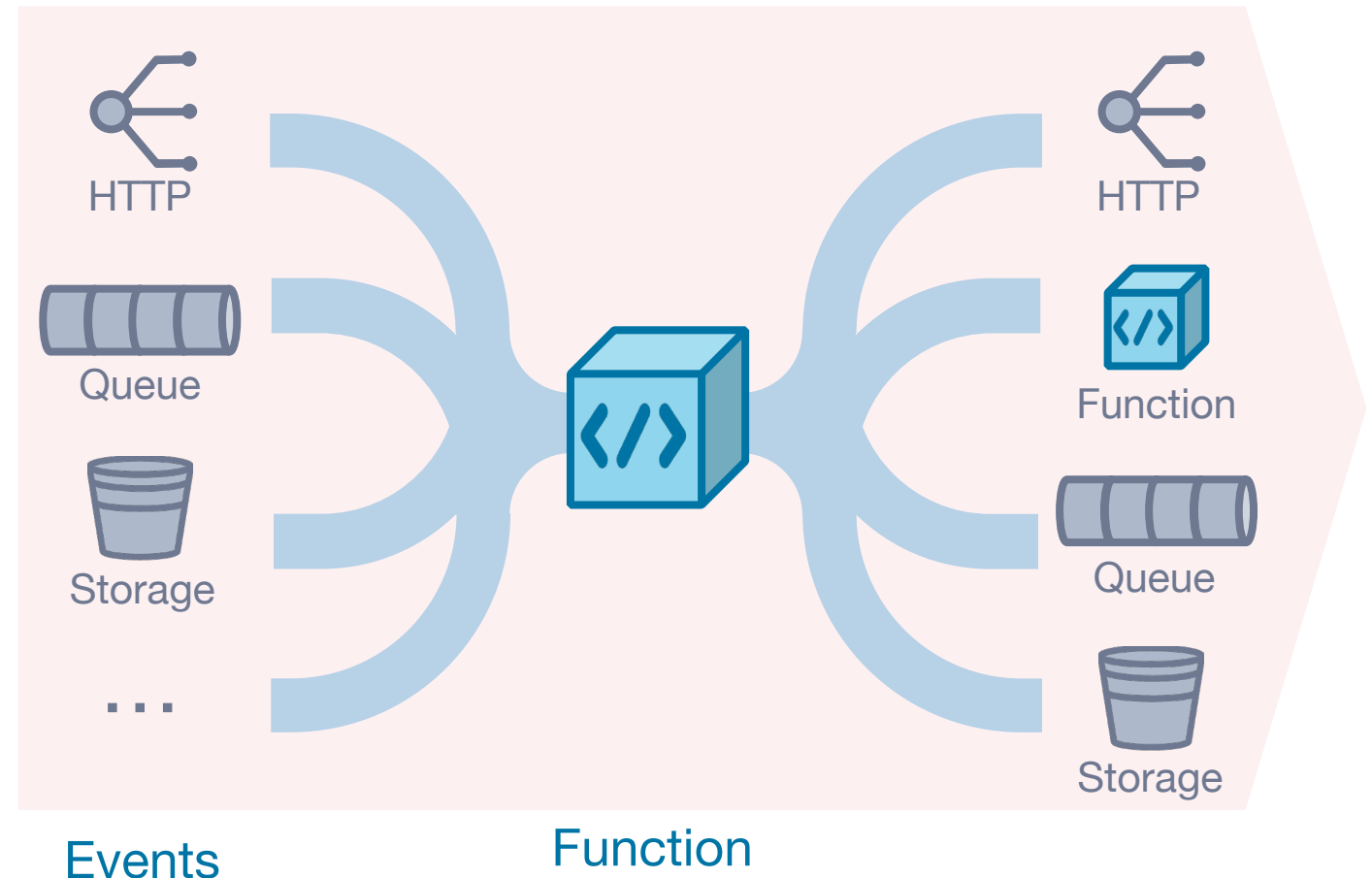
Prof. Stefan Tai
Sebastian Werner, Elias Grünewald, Nicola Leschke, Maria Borges

Information Systems Engineering
TU Berlin

ReCap: Function-as-a-Service (FaaS)

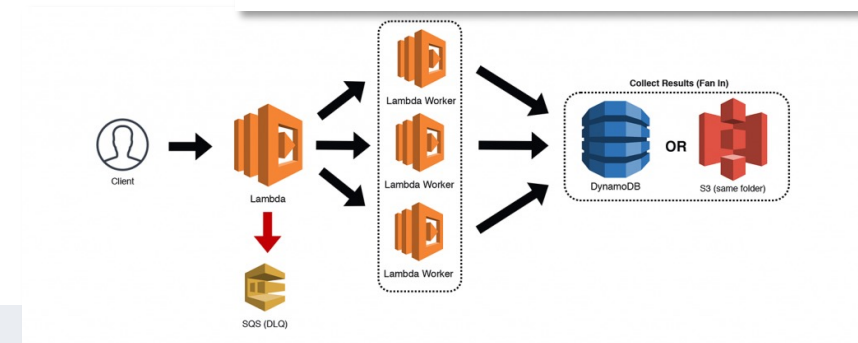
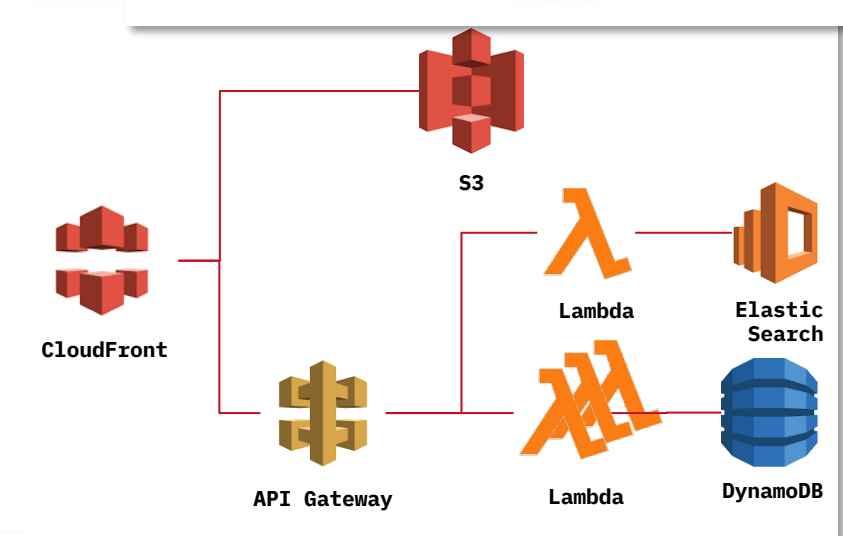
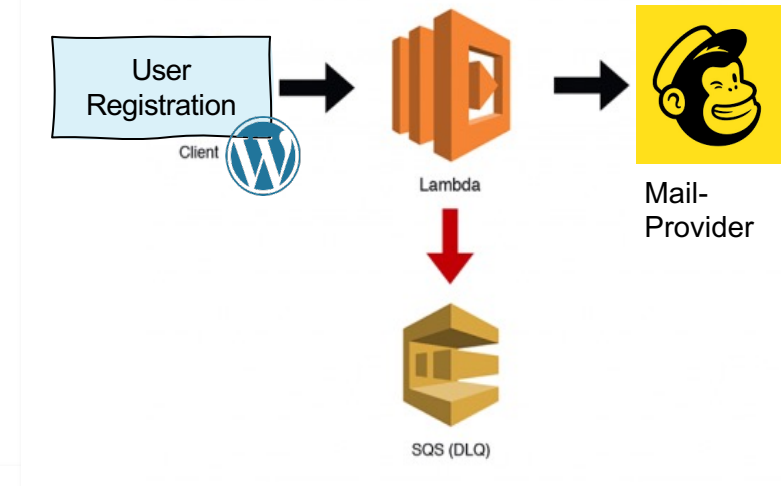
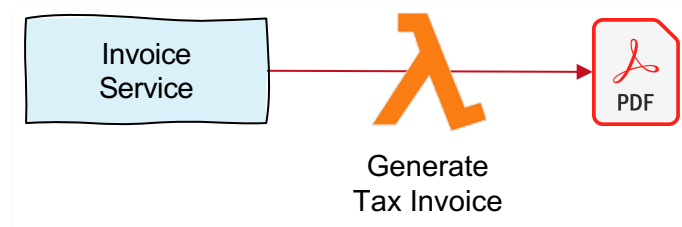
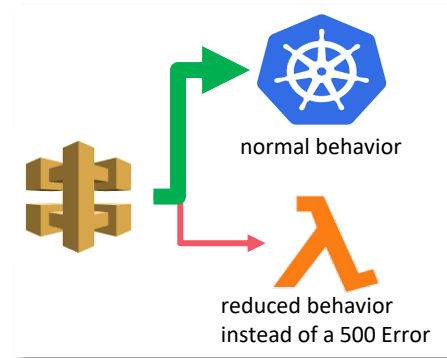
- **Generic** user-defined managed execution
- execution-based billing model
- Provider-side **managed operational tasks**
- Workload-driven **elastic autoscaling**
- Stateless and ephemeral

Event-based programming model



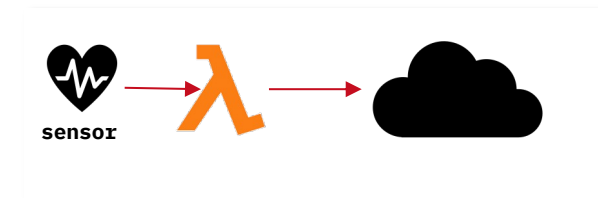
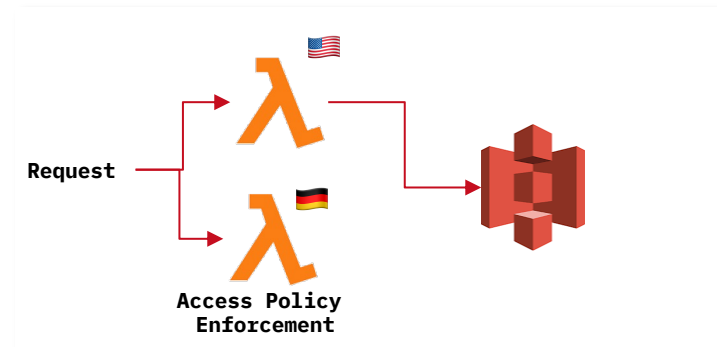
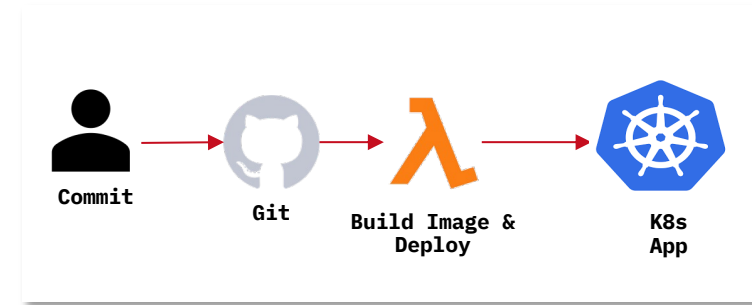
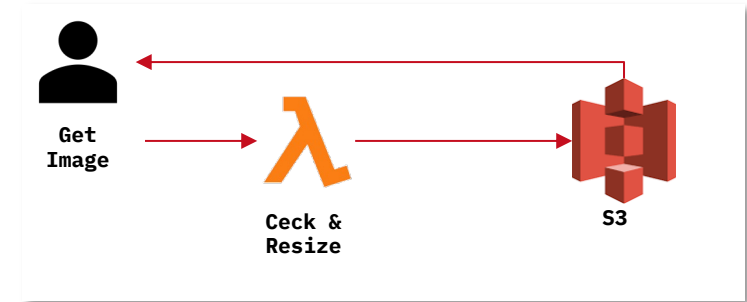
ReCap: When and How you can use FaaS

- Scalable Glue-Code
- Infrequent Tasks
- Web-Applications
- Failover Handlers
- Processing Jobs



ReCap: When and How you can use FaaS

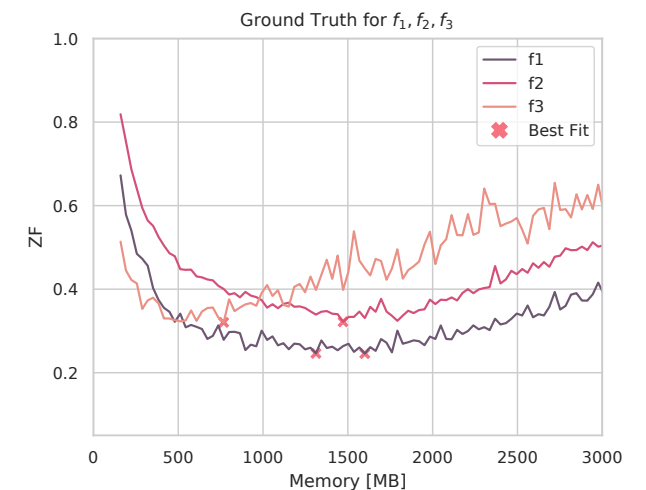
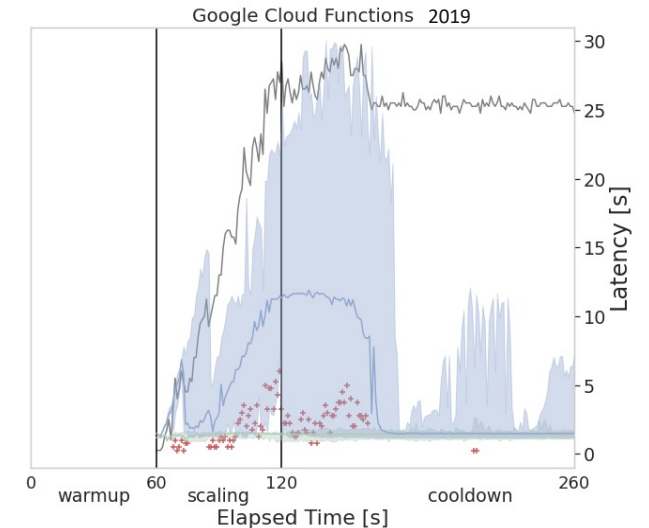
- Scalable Glue-Code
- Infrequent Tasks
- Web-Applications
- Failover Handlers
- Processing Jobs
- **Cloud Automation Code**
- **Serverless-Edge Computing**



1. What is serverless?
2. Patterns in FaaS-based Applications
3. When and How you can use FaaS
- 4. Pitfalls and Words of Caution**
5. A Comprehensive Example

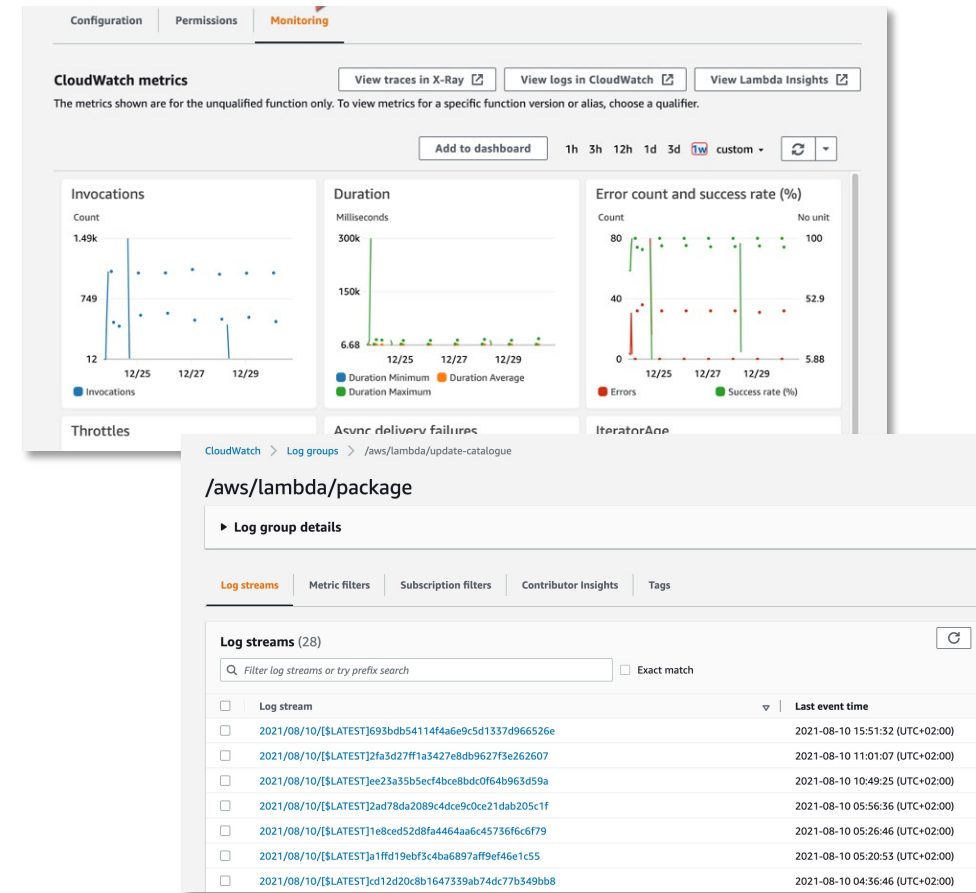
Less Operational Tasks = Less Control

- When you deploy a FaaS application, you define:
 - how many functions should at most run in parallel
 - how many functions should be running all the time
- Horizontal Scaling:
 - up to the FaaS platform, might not be as you expect
 - be wary of cold-starts
- Vertical Scaling:
 - few tuning knobs, e.g., on GCF only memory
 - highly sensitive, changes in memory affect CPU, Memory, IO, Network



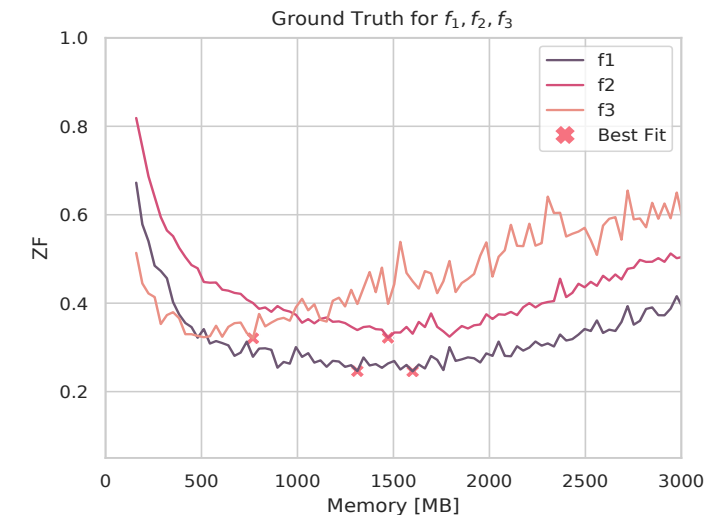
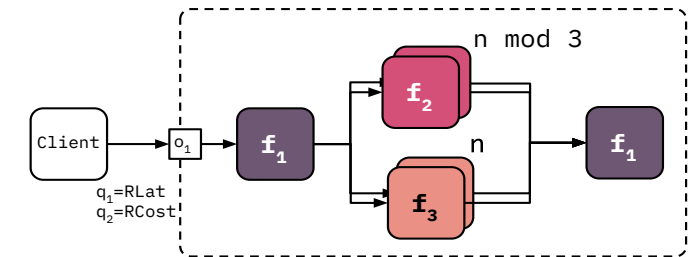
Observability

- A fully-managed monitoring and logging solution is typically offered by the FaaS provider (at an additional cost..)
- e.g. AWS CloudWatch Metrics:
 - # of invocations
 - Concurrent executions
 - Duration of functions
 - Error count / success rate
 - Etc..
 - (Custom metrics can also be added to each function)
- Vendor-specific tooling, with vendor choices for what is important to observe
- Whenever you use external services observability might get “thin”



Cost

- Highly sensitive configuration knobs, e.g., memory impact may runtime capabilities
- On AWS, every 1500MB memory add 1 VCPU and 0.5 Gbps network throughput → selecting the right memory influences execution time even if the function dose not need the memory
- You need to understand the relationship between functions under different workload scenarios to size them properly
- Cost performance tradeoffs (especially for complex compositions), high chance of oversizing

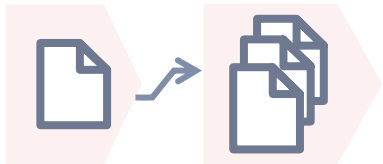


FaaS: Assumptions and Realities



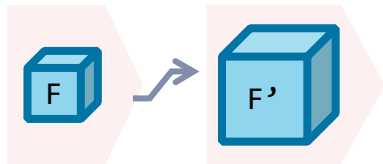
Function abstraction matches all workload.

Workload must match function handler configurations.



Functions scale automatically with incoming events.

Provisioning time may not suit scaling app needs.



Fully managed ops tasks solve diverse ops needs.

Fixed strategies for applying ops tasks.



Functions can easily be composed to implement any business logic.

Function compositions experience complex limitations and add costs.



FaaS offers significant advantages for execution cost.

Marginal execution costs are higher, and compositions are cost drivers.

A word of caution...

“Microservices and serverless components are tools that do work at high scale, but whether to use them over monolith has to be made on a case-by-case basis.”

Marcin Kolny – AWS Prime Team

<https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90>

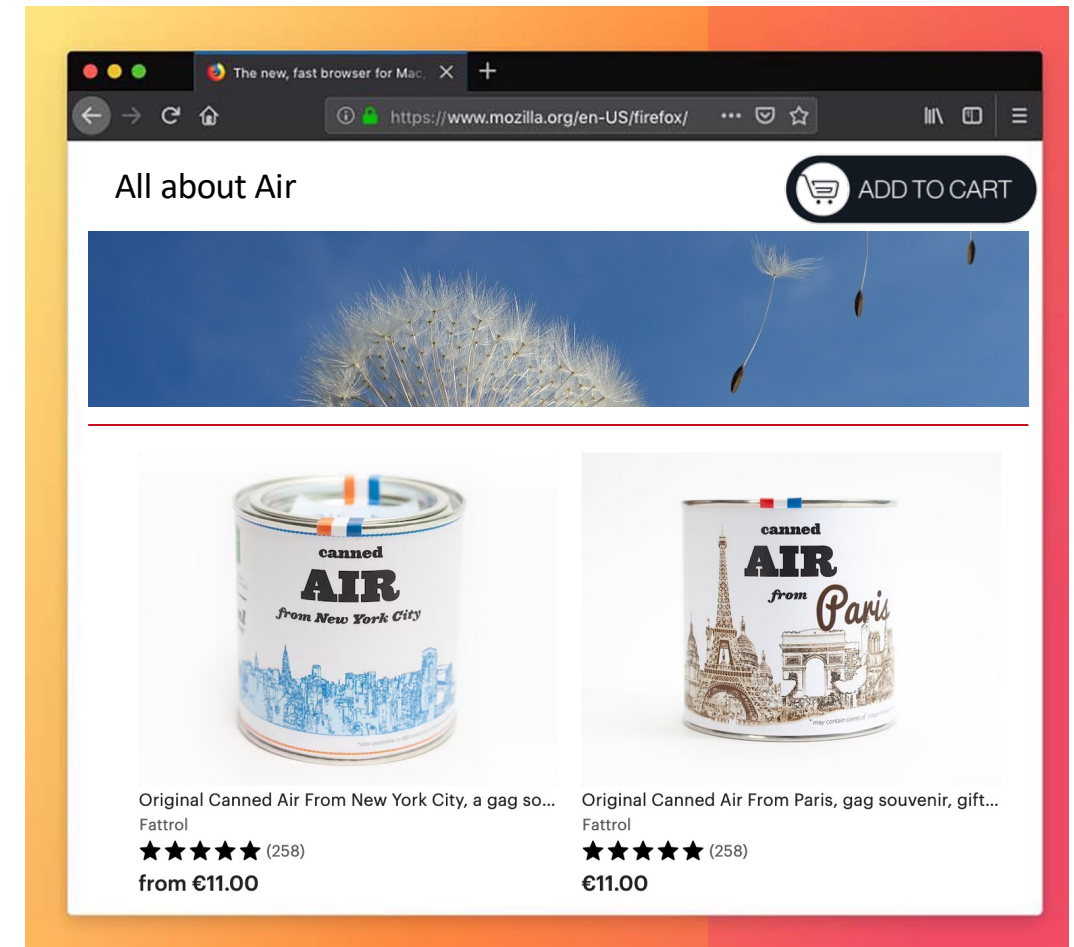
1. What is serverless?
2. Patterns in FaaS-based Applications
3. When and How you can use FaaS
4. Pitfalls and Words of Caution
- 5. A Comprehensive Example**

Build a Webstore – “CanAir”

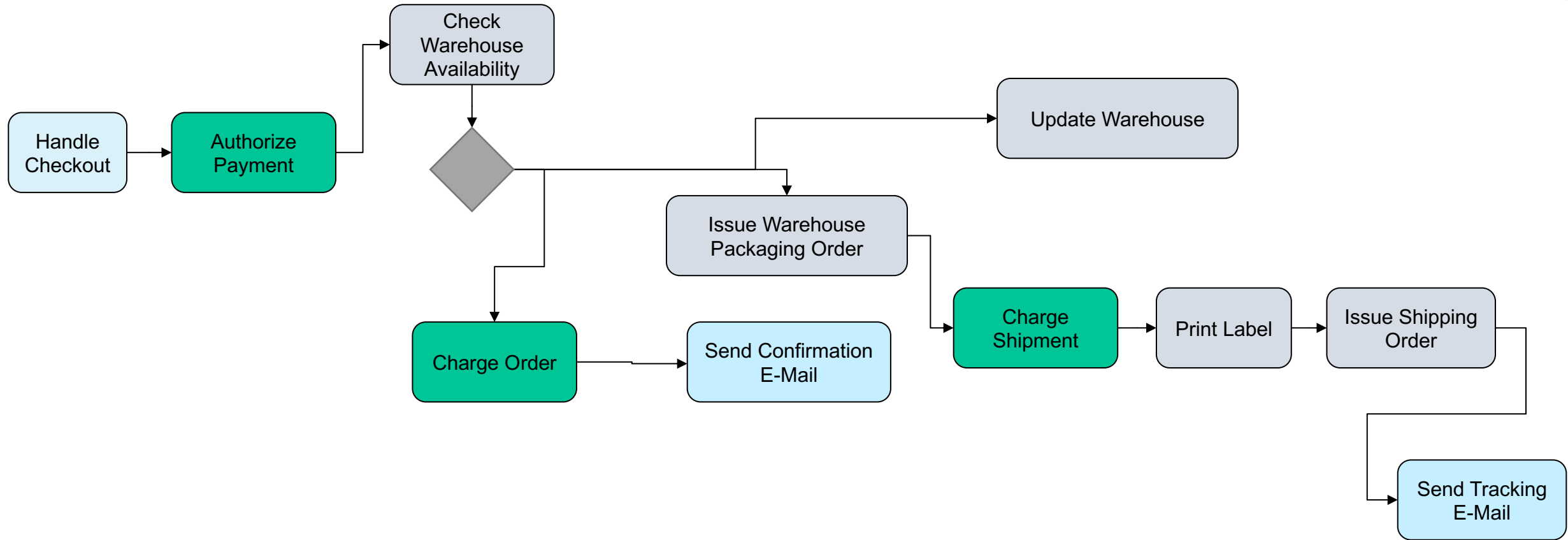
Scenario:

We want to sell quality AIR™ to customers.
We already have the website and
warehouse ready to go.
Our product will soon be featured in Shark
Tank (“Die Höhle der Löwen”)

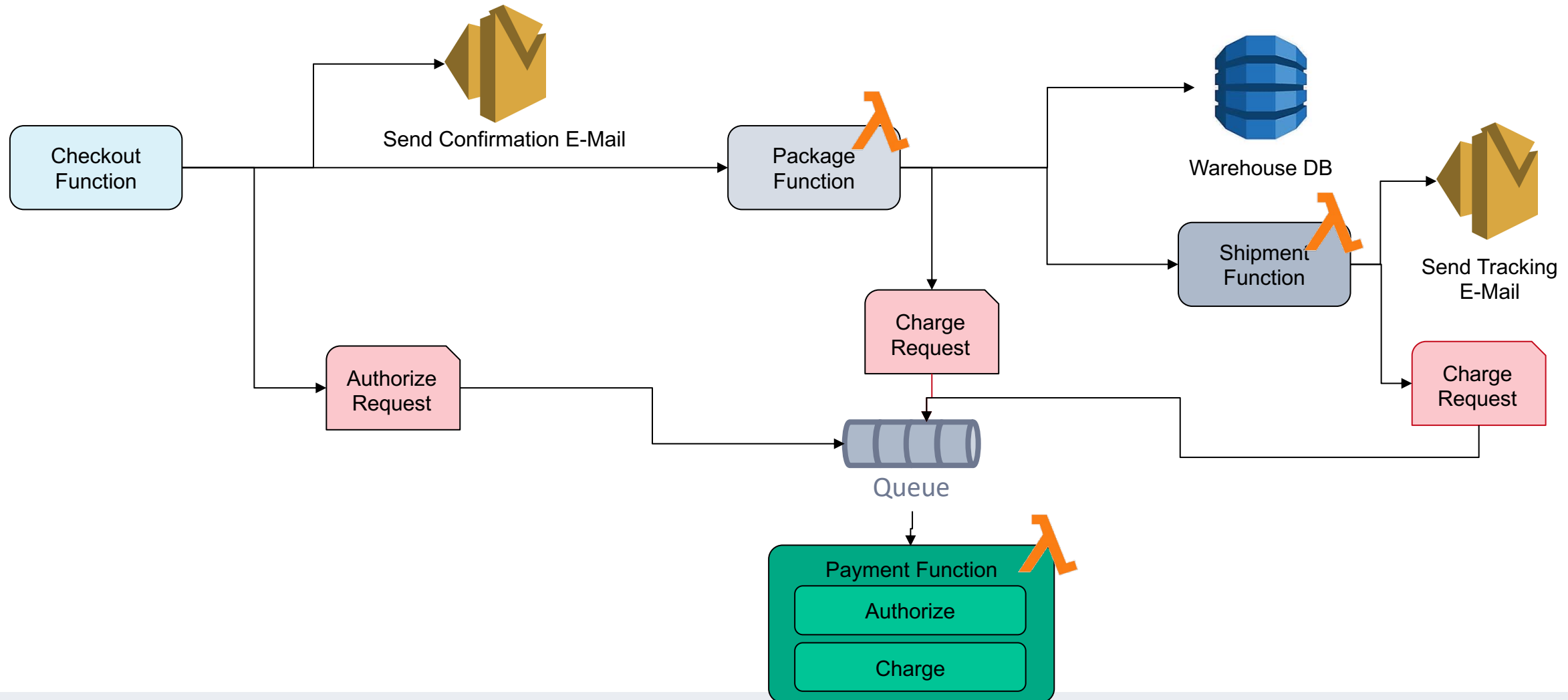
How dose a serverless solutions for AIR™
look like?



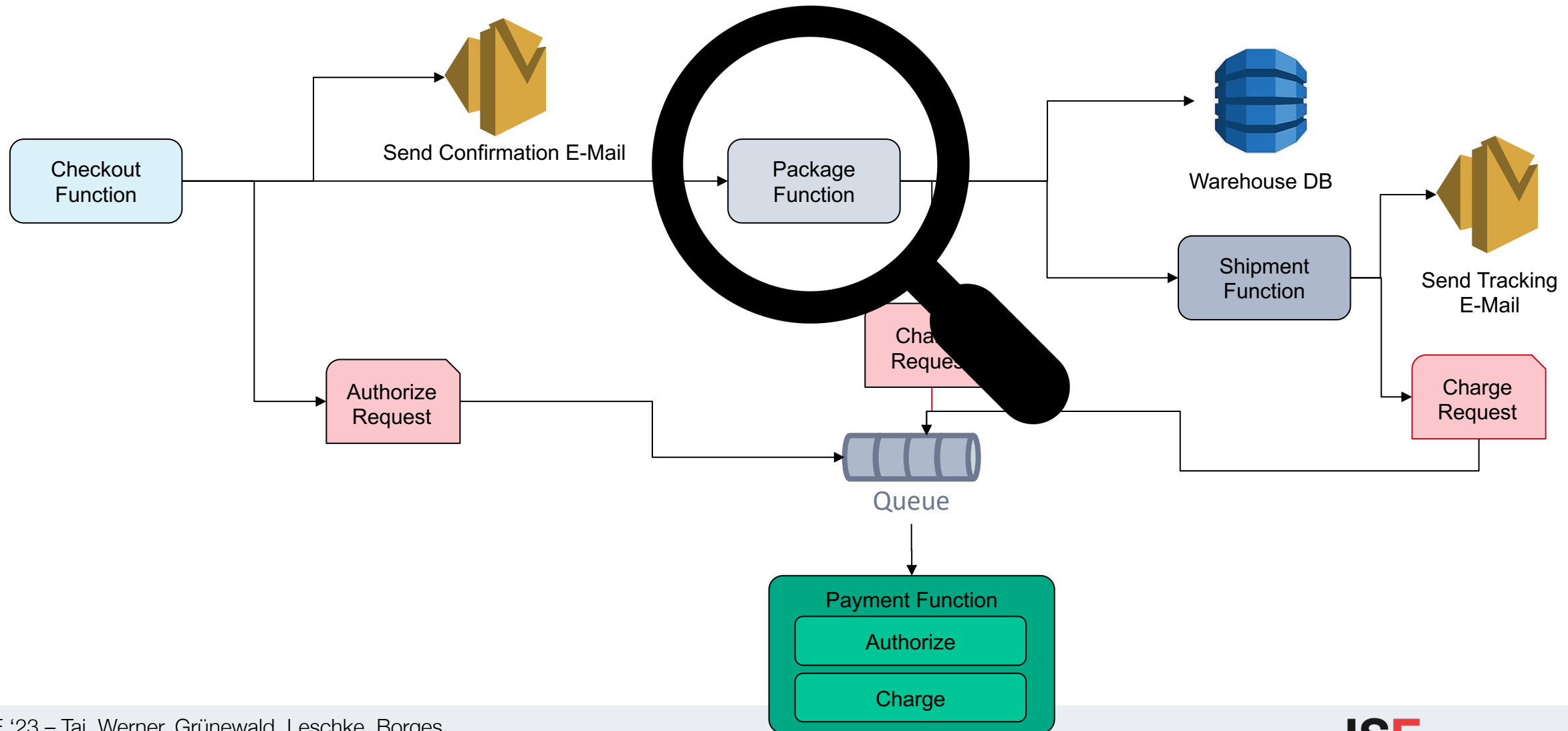
Checkout Process



The Serverless way - Architecture



The Serverless way - Architecture



Package-Function



```
DB = connect()

def handler (event, context):
    for item in event.order:
        if checkAvailability(DB, item.id,item.quantity):
            emitChargeRequest(item,event.account)
            emitPackageOrder(item)
            updateItemQuantity(DB,item.id, -1*item.quantity)
        else:
            error(item,event.order)
    callShipmentFunc(event.order,event.account)
    return dict(status:"ok")
```

package.py

AirShopCommon (4MB)

Request (3.4kB)

boto3 (1.2kB)

```
{
  „timeout“:30,
  „memory“:256,
  „concurrent-limit“:1000,
  „runtime“:“python3.7“
}
```

package_config.yaml

Testing

- Always a composition
 - can only be tested in composition
 - composition => other services we need to pay for
- Local service replication, e.g., localstack for AWS



LocalStack is a cloud service emulator that runs in a single container on your laptop or in your CI environment. With LocalStack, you can run your AWS applications or Lambdas entirely on your local machine without connecting to a remote cloud provider!

Deployment

- Many functions
 - many "moving parts", e.g., layers, code, events
 - many configurations, e.g., IAM, environment variables, ...
- IaC for serverless applications, e.g., serverless framework, aws cdk, SAM, ...

```
1  type: image-processing-api
2
3  inputTypes:
4    s3Bucket:
5      type: string
6      required: true
7      description: The name of the S3 bucket which contains the
8
9  components:
10    filesBucket:
11      type: aws-s3-bucket
12      inputs:
13        name: user-profile-images
14    apiEndpoint:
15      type: rest-api
16      inputs:
17        gateway: aws-apigateway
18        routes:
19          /process:
20            post:
21              function: ${imageProcessorFunction}
22              cors: true
23    imageProcessorFunction:
24      type: aws-lambda
25      inputs:
26        handler: index.process
```

serverless  framework

Recommended tooling

Serverless Framework

<https://www.serverless.com/>

AWS Lambda Tutorials

<https://aws.amazon.com/training/learn-about/serverless/>

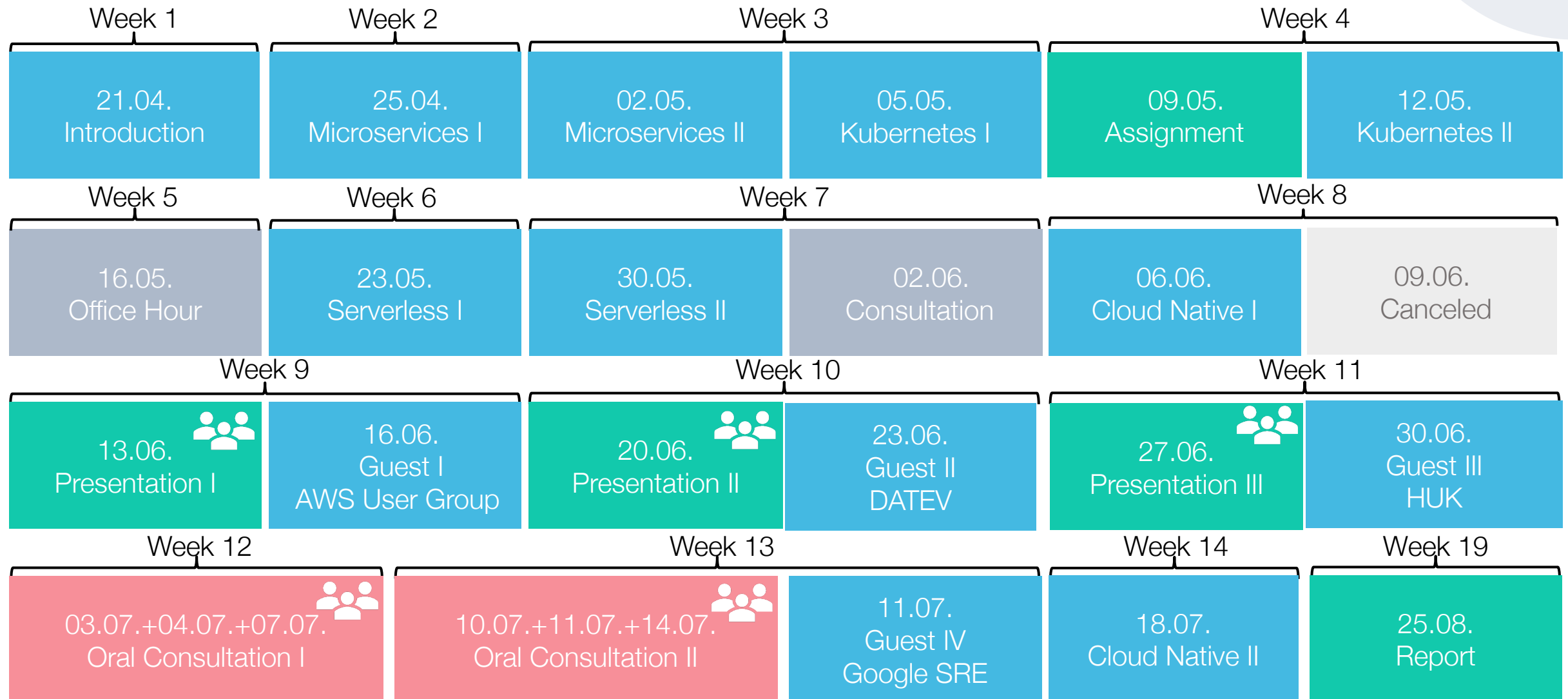
Azure Serverless Cook Book

<https://azure.microsoft.com/en-us/resources/azure-serverless-computing-cookbook/>

IBM Serverless Intro

<https://www.ibm.com/topics/serverless>

Schedule



Code of the lambda example



```
import boto3
from botocore.errorfactory import ClientError

import PIL
from PIL import Image
from io import BytesIO
import os
import logging as log

#set the global variables
BUCKET = os.environ["BUCKET"]
REGION = os.environ["AWS_REGION"]

S3_ENDPOINT_URL = os.environ.get("S3_ENDPOINT_URL")
S3_KEY = os.environ.get("S3_KEY")
S3_SECRET = os.environ.get("S3_SECRET")

s3 = None
if S3_ENDPOINT_URL and S3_KEY and S3_SECRET:
    s3 = boto3.resource(
        's3',
        endpoint_url=S3_ENDPOINT_URL,
        aws_access_key_id=S3_KEY,
        aws_secret_access_key=S3_SECRET,
    )
    log.info("Using custom endpoint configuration for S3")
else:
    s3 = boto3.resource('s3')
    log.info("Using default endpoint configuration for S3")

if s3 is None:
    raise Exception("Failed to initialize S3 client")

def call(event, context):
    key = event["pathParameters"]["image"]
    size = event["pathParameters"]["size"]

    result_url = resize_image(BUCKET, key, size)

    response = {
        "statusCode": 301,
        "body": "",
        "headers": {
            "location": result_url
        }
    }
```

```
    }
}

return response

def resize_image(bucket_name, key, size):
    resized_key="{size}_{key}".format(size=size, key=key)
    bucket = s3.Bucket(bucket_name)

    objs = list(bucket.objects.filter(Prefix=resized_key))
    keys = set(i.key for i in objs)
    if resized_key in keys:
        return resized_image_url(resized_key, bucket_name, REGION)
    else:
        # the object does not exist, time to resize
        obj = s3.Object(
            bucket_name=bucket_name,
            key=key,
        )
        obj_body = obj.get()['Body'].read()

        size_split = size.split('x')
        img = Image.open(BytesIO(obj_body))
        img = img.resize((int(size_split[0]), int(size_split[1])), PIL.Image.ANTIALIAS)
        buffer = BytesIO()
        img.save(buffer, 'JPEG')
        buffer.seek(0)

        obj = s3.Object(
            bucket_name=bucket_name,
            key=resized_key,
        )
        obj.put(Body=buffer, ContentType='image/jpeg')
        object_acl = s3.ObjectAcl(bucket_name, resized_key)
        object_acl.put(ACL='public-read')
        return resized_image_url(resized_key, bucket_name, REGION)

def resized_image_url(resized_key, bucket, region):
    return "https://s3-{region}.amazonaws.com/{bucket}/{resized_key}".format(bucket=bucket, region=region,
    resized_key=resized_key)
```